Distributed Data Management and Integration: The Mobius Project

Shannon Hastings and Stephen Langella, Scott Oster, Joel Saltz

Department of Biomedical Informatics
Multiscale Computing Laboratory
The Ohio State University
Columbus, OH, 43210

Abstract. Advances in computational resources, the concept of *the Grid* and the development of middleware and applications to build and support the Grid will give Grid users the power to access and interpret large amounts of heterogeneous data. *Metadata*, data that describes data, can be used to represent and define the protocols of the Grid, as an abstraction between services and datasets, and to provide syntactic or semantic descriptions and annotations of any sized datasets so that they may be discovered, communicated, and analyzed efficiently. In this document, we will present Mobius, which can be described as a generic and extensible set of protocols and services for managing data in a Grid environment.

1 Introduction

With the emergence of Grid computing and Web services architectures, it is increasingly critical to address the information service needs of loosely coupled systems. A multi-institutional Grid environment will contain many data sources, which maintain different types of data using different storage mechanisms. Management of data and metadata is a major task in such systems. Seamless integration of data can facilitate the discovery of data-oriented findings that would not be possible without a system that supports the distributed management of metadata and data in a scalable environment. A middleware system which provides basic building block services for metadata and data management can be extended by many other service areas such as semantic query services, ad-hoc data warehouse services, and specialized data integration services.

In this paper, we describe the architecture of Mobius, a middleware framework designed for efficient management of data and metadata in dynamic, distributed environments. Its design is motivated by the Grid [7, 5, 6] (in particular by the activities of the Data Access and Integration Services group at Global Grid Forum [8, 4, 1, 2]), by earlier work done at General Electric's Global Research Center [13], and by particular domain application area requirements. Biomedical research studies, for example, can involve integration of data, including proteomic, molecular, genomic, and image data in a multi-institutional environment. Mobius provides a set of generic services and protocols to support distributed creation, versioning, and management of data models and data instances, on demand creation of databases, federation of existing databases, and querying of data in a distributed environment. Its services employ XML schemas to represent metadata defi nitions (data models) and XML documents to represent and exchange data instances.

As an example, consider biomedical research studies that collect data in complex data types, with partial syntactic and semantic overlap. A researcher can develop a hypothesis and accrue several types of patient and laboratory related data. The researcher needs to create databases to maintain data for patients and laboratory results. Data may have also been previously stored in multiple sources and databases, potentially created by other researchers. In which case, data must be integrated from these potentially heterogeneous sources for analysis. The researcher should be able to use a system to create an ad-hoc data warehouse spanning multiple databases (2D gel data, clinical data, lab data, drug treatment data, and molecular data) to enable distributed analysis. The researcher should also be able to use the ad-hoc data warehouse to carry out queries to test hypotheses. Any two databases may defi ne data that contain the same semantic content with completely different structure representations. The analysis of data may lead to collection of new datasets as well as new types of data. This type of scenario can be supported by a system that will allow the researcher to:

- create schemas which describe their data models, register and share these data models with other collaborators, manage and version them while new data types are added or deleted,
- facilitate translation between data models that have the same semantic meaning, but different structure, and
- create, integrate, and manage databases and data that conform to these data models.

Mobius consists of three core service areas: Global Model Exchange (GME), Data Instance Management (Mako), and Data Translation Service (DTS). These service areas and underlying protocols support distributed creation, versioning, management of data models defi ned by XML schema, on-demand creation of distributed databases, federation of existing databases, querying of data in a distributed environment, and on demand translation of instance data from one data model to another.

Using Mobius, the biomedical researcher can develop databases and querying capabilities for her studies as follows. The researcher fi rst designs XML schemas describing the data types she wants to maintain. The GME provides several alternatives for the researcher to create and register her schemas: 1) The researcher can search the GME for existing schemas and may use one that suits her research study. 2) She can version an existing schema by adding or deleting data attributes. 4) The researcher can create a new schema with new attributes and structure. 5) She can compose a schema using new attributes and by referencing multiple existing schema entities in her schema. Once the schema is created, it is registered with the GME so that other researchers can search for it, and the Mako services can use it to create new databases and validate data against its data model. The researcher can now instantiate one or more Mako servers to maintain the databases conforming to the schemas. She can also register the schemas with existing running Mako servers. The Mako servers create databases using the XML schemas so that new data can be entered and maintained across the system. When a new data set is submitted (as an XML document) to a Mako server, the server ingests the document, stores the data specifi ed in the document in the databases, and indexes them. Any given data set can also be distributed across a collection of Mako servers and rematerialized as needed at query or retrieval time. With the data effectively stored, the researcher can then retrieve data from these databases using queries expressed in XPath [3].

2 Requirements

As mentioned earlier, the world is increasingly becoming more dependent on electronic data management from patient medical records and research data spread across institutions around the world to remote monitoring of airline engine sensor data. Data is everywhere, and the easier that it can be integrated and used together the more useful it becomes. This section will enumerate some basic requirements for managing data in a grid environment and elaborate on each in a use case driven approach.

2.1 Global Model Management

In order for services on the grid to communicate with each other, their data must be described in a format that is understood by each involved component. Thus a data management system for the grid must provide a method for defining metadata and data, we will refer to this definition as a data model. A data model is the specification of the structure, format, syntax, and occurrences of data instances. In order for services to communicate with one another they must agree on a data model over which they communicate. In order for such models to exist, they must be globally available to every service, assuming the service has access control rights to obtain and view the model. Making data models globally available and uniquely identifiable forms a conceptual global model consisting of the individual models. Therefore, given proper credentials, one model may reference previously defined entities in another model. For example, if the entity *Patient* is defined in the model *Hospital*, it could be referenced by the model *Clinic*. This means that the Clinic's definition of Patient is equivalent to the Hospital's definition of Patient. Since all entities are globally identifiable, entities within the schema must be unique. Therefore once the Patient entity is defined, it cannot be defined by another model. Although this promotes standardization of data definitions, this will not be an acceptable solution for the grid, as there will invariably be competing definitions for these entities. This problem can be solved by namespaces. Entities defined in models should be assigned to a namespace, where entities are unique within their namespace. The combination of the entities name and namespace makes the entity unique within the grid. Thus, entities can be referenced, systems and data integration become simpler, and standards can be promoted.

2.2 Data Instance Management

Services on the grid will need a method for storing and querying data and metadata. This will be integral for service communication on the grid. Given valid credentials, data should be able to be stored across a series of heterogeneous loosely-coupled machines. This will allow for data storage systems to be virtually any size and scale within the grid as well as become ad-hoc and dynamic in size and shape. The system should also provide the ability to allow trusted users to update, query and delete the data. These generic base storage service interfaces can be extended by application specific grid users in order to tailor them for specific needs.

2.3 Data Translation

It will often be the case that institutions will have different methods for modeling data which may be conceptually or semantically the same. It would be beneficial to both institutions to be able to translate between one another's data models such that each can leverage the other services. Where possible, the simplest way to share data and services is to use common models, but in practice, this is not always possible. This begs the existence of a registry architecture for managing and discovering services on the grid that translate between defined data models. Such an architecture would allow services to programmatically identify and leverage other computational services which would otherwise not be possible. This, of course, does not mean that the automatic structural or semantic data translation problem is solved. This simply gives a service architecture where data translation services, possibly written by domain experts, are published and can be trusted between the two domains in which the data is translated and potentially accepted by others as the standard.

3 Related Work

Due to the nature of a large componentized middleware architecture, covering all possible related work which may be leveraged by this architecture is beyond the scope of this paper. In this section we will cover the related work which deals specifically with grid middleware protocols and grid metadata management; the main contributing components of the mobius architecture. We will not cover related work pertaining to specific service implementation details. However, if there is related work which addresses a similar problem, as a Mobius service implementation, it can simply be wrapped with the Mobius service protocols or be used in coordination with the Mobius middleware.

There have been several projects which focus particularly on service infrastructure for metadata management, and a few which specifically target execution on the grid. Storage Resource Broker (SRB) [20] in coordination with the Meta Information Catalogue (MCAT) [15] is a grid-enabled middleware system for accessing heterogeneous data storage services in a client-server framework. In its current incarnation, the SRB system uses the term metadata to mean a predetermined set of key-value pairs which describe attributes of the data set. The user can add extra key-value entities which can be used to query and discover data sets in the SRB framework. The key defining difference between the SRB/MCAT solution and the Mobius framework is the way in which metadata is handled in the environment. The metadata in SRB is not a user-designed structure which is a published and managed entity. It is used to describe a specific data set or data sets and the metadata itself does not contain a particular unique name, or structure (other than key-value). The Mobius framework enables structured metadata of any size or shape to be user-defined, published, and managed. It also enables instance data conforming to this data to be created and validated. Although the two systems at a high level seem to support similar interactions, store and retrieve data which can be queried and discovered using metadata, they are quite different in scope and direction.

Another grid system for management of metadata is the Metadata Catalog Service (MCS) [16]. The MCS system was originally created to support metadata management in the Grid Physics Network (GryPhyN) [10]. MCS is closely related to SRB in that its current use and aim is to store metadata about logical files in a data grid. Although it does not provide standard data access interfaces that SRB or Mobius provides, it does store and allow querying of these logical files' metadata. Like MCAT, MCS models metadata as a set key-value pairs and not as a user-defined complex structure which can be comprised of other user-defined structures. Mobius addresses the concern that a generic platform for metadata in a grid should be capable of being structured and comprised of pre-existing structures from experts of their respective domains.

4 Global Model Exchange

The driving force behind the need for a global service architecture to manage user-defined data models can be laid out as a set of use-case driven requirements. A user in the grid would like to be able to create, publish, and version a data model, possibly defined using XML Schema. That model may be used by many other services at service runtime. This feature requires that those requesting services be able to request model definitions by namespace and name, and be guaranteed that they will receive the same model as any other service issuing the same request. The publishing grid user would like other grid services to be able to use that model and detect and use its changes, enabling possible programmatic updating of running services. A particular model may be made up of several sub-models as defined under different namespaces. The grid user would like to be assured that when a model is referenced in the grid, there is some assurance it is always available, as well as any of the sub-models that it may be referencing which are defined by other grid users. A service infrastructure that provides an implementation of these requirements would be the core building block for other services as model-to-model translation or mapping services, generic ad-hoc model instance storage services, and robust service-to-service data communication. These requirements could mostly be met with a combination of current technologies and with a large amount of new philosophic agreements with all users. For example, the current defacto standard for schema publishing uses web servers, HTTP, and DNS maintained namespaces, and schemas are downloaded via HTTP requests to a URL matching their fully qualified names. If a certain level of service and availability could be guaranteed, servers maintained all pervious versions, and authorities were put in place to allow users to publish schemas to appropriate namespaces, we could approach a solution to these problems. However, a new set of grid services with the sole purpose of providing an implementation of these requirements will be much easier to adopt in practice, and will be a fundamental building block of many future data driven grid services.

The Global Model Exchange (GME) is a service that responds to the requirements stated above. It is responsible for storing and linking data models as defined inside namespaces in the distributed environment. The GME enables other services to publish, retrieve, discover, remove, and version metadata definitions. The GME services are

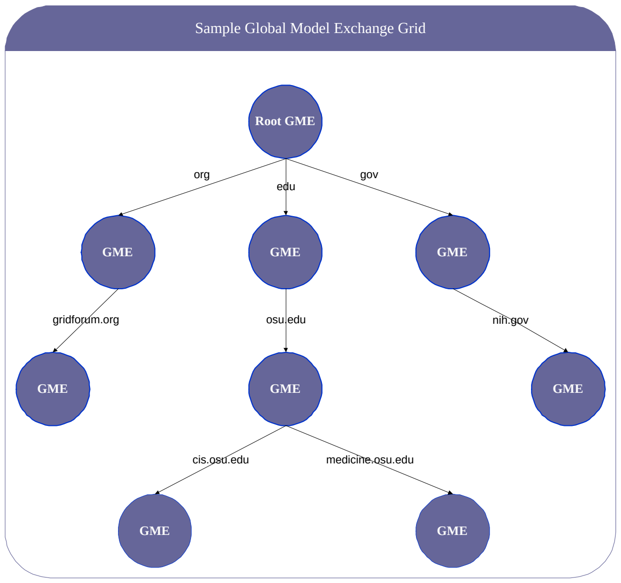

Fig. 1. GME Architecture

composed together in a DNS-like architecture representing a parent-child namespace hierarchy as shown in the figure 1.

The *GMEAccess* interface is the user-level interface that exposes the functionality of these protocol messages through a basic set of service access methods. A user service can use the GMEAccess interface to publish a model, request an instance of a particular version of a model, and post a query against the models of a particular GME namespace.

The GME provides model version and model to model dependency management. For instance, if a user service publishes a model to the GME, and later the model is modified and republished, the model will automatically be versioned and both models can be used concurrently. Furthermore, if a suitable translation mapping is provided for changes between the two versions, they can be interchanged seamlessly. The protocol provides a mechanism for stating the exact version of a model for a given request. A model can also contain types defined by other models or references to types contained in other models. This reference integrity might be considered the largest requirement for a GME that the current use of a URL does not provide. The role of the GME in the greater picture is to ensure distributed model evolution and integrity while providing the ability for storage, retrieval, versioning, and discovery of models of all shape, complexity, and interconnectedness in a grid-like environment.

A future extension of the GME service architecture would be to support semantic model storage, versioning, and querying. Storing data models without semantics is a base case building block for being able to begin to store, search, and reason about semantic data models. We would like to extend the GME basic service by not only storing the model, but also adopting a semantic model definition language such as RDF [18] and provide higher level querying for those models. One could imagine being able to pose the question, ”Are there any models published anywhere in the grid that have something to do with cancer research?”. With higher level semantic model storage and querying, questions like this can be applied across all data models in all namespaces in the grid.

5 Mako

Mako is a service that exposes data resources as XML data services through a set of well-defined interfaces based on the Mako protocol. A data resource can be a relational database, an XML database, a file system, or any other data source. Data resources are exposed through a set of well-defined interfaces, thus exposing specific data resource operations as XML operations. For example, once exposed, a relational database would be queried through Mako using XPath as opposed to querying it directly with SQL. Mako provides a standard way of interacting with data resources, thus making it easy for applications to interact with heterogeneous data resources.

5.1 Mako Interfaces

The Mako client interfaces are similar to those of an XML database, however, since Mako is a distributed service, the interfaces themselves are motivated by the work of the Data Access and Integration (DAIS) [4, 1, 2] working group in the Global Grid Forum (GGF) [8]. Mako defines a set of access interfaces that define the data operations, and it also defines a set of handles that implement said operations on a given context. For example, the XPathAccess interface which defines XPath operations, is implemented by the XMLCollectionHandle and by the XMLDocumentHandle. In the context of the XMLCollectionHandle the XPath operations are implemented and applied across the entire collection, whereas in the context of the XMLDocumentHandle the XPath operations are performed only against the XML document that the XMLDocumentHandle represents. In all Mako provides three handle types, a XML Data Service Handle, a XML Collection Handle, and a XML Document Handle.

5.2 Mako Architecture

Clients interact with Mako over a network; the Mako architecture illustrated in Figure 2 contains a set of listeners, each using an implementation of the Mako communication interface. The Mako communication interface allows clients to communicate with a Mako using any communication protocol. For example, if the communication interface is implemented using Globus [9] security, clients may communicate with Mako using the Globus Security Infrastructure (GSI). Each Mako can be configured with a set of listeners, where each listener communicates using a specified communication interface.

When a listener receives a packet, the packet is materialized and passed to a packet router. The packet router determines the type of packet and decides if it has a handler, described below, for processing a packet of that type. If such a handler exists, the packet is passed onto the handler which processes the packet and sends a response to the client.

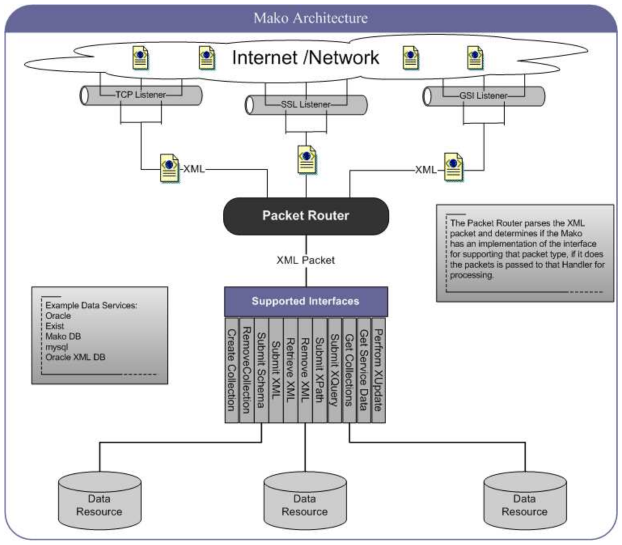

Fig. 2. Mako Architecture

5.3 Mako Protocol

The Mako Protocol defines a set the packet types that a Mako may process. This section will give an overview of some of the operations that the Mako protocol supports.

Service Data Service Data, a term often used by the OGSI [17] working group in the GGF, refers to metadata about a given service. Mako service data contains information on when the Mako was started, information on the underlying data resource, a list of request types supported, etc.

Collection Management In the context of XML databases, the common nomenclature for referring to a group of related instance documents in a single storage service is a Collection. This is very similar to the relational database concept of a database. The concept diverges slightly, in that collections can have sub collections, and collections may not have a single schema (or any at all) associated with them. Mako has three request packets for managing collections of XML documents. The CreateCollectionRequest packet is used to create a new XML collection on a Mako. The RemoveCollectionRequest packet can be used to remove a collection from a Mako. Finally, the CollectionListRequest packet requests a list of collections that exists on a Mako or in a sub collections on a Mako.

Schema Management Each collection in Mako can be restricted to only accept XML documents from a set of certain types. This is accomplished by specifying a set of XML schemas in which a XML document must be validated against. The Mako protocol provides a method for adding and removing schemas to and from an XML collection within a Mako. The SubmitSchemaRequest packet is used to submit schemas to a collection in a Mako, and the RemoveSchemaRequest packet removes them. The SchemaListRequest packet can be used to get the list of schemas supported by an XML collection.

Document Management The Mako protocol defines methods for submitting, updating, removing, and retrieving XML documents. The SubmitXMLRequest packet is used to submit documents to an XML collection in a Mako. Upon submission, Mako assigns each entity a unique identifier; later we will see why this identifier is important. XML documents that reside in a Mako can be updated using XUpdate [14], the XUpdateRequest packet is used to accomplish this.

Mako provides two methods of removing documents. First, a list of documents can be removed by specifying their unique identifiers, this is done using the RemoveXML-Request packet. Second, XML documents can be removed by specifying an element identifying XPath [3] expression. The XPathRemoveRequest packet is used to remove XML documents that meet an XPath expression.

The RetrieveXMLRequest packet is used to retrieve XML documents, or a subset of XML documents, from a Mako. Documents, or subsets of XML documents, can be retrieved by specifying their unique identifier. Recall that each element in an XML document is given an identifier, making any subset of a document uniquely addressable. The Mako protocol also allows the level of retrieval to be specified. For example, if you think of an XML document as a tree, then given a unique identifier, one would be able to specify how many levels of children should be included in the materialization of the document. Elements containing children that are below the height specified would not be included in the materialization, however references to their immediate children would be included. This feature becomes quite valuable when working with large datasets, in that full documents do not need to be materialized just to view partial results. It also allows one to build a demand driven Document Object Model (DOM) on top of the protocol. In general such a feature improves application performance by allowing the application to specify how data is materialized.

Querying Mako provides query support through the use of XPath [3], XPath queries are performed using the XPathRequest packet.

5.4 Mako Handlers

As specified in the architecture section, when a Mako receives a packet, the packet is given to the handler that processes that type of packet. Thus, for each packet type in the Mako protocol, there is a corresponding handler to process that packet type. In order to abstract the Mako infrastructure from the underlying data resource, there is an abstract handler for each packet type. Thus, for a given data resource, a handler extending the abstract handler would be created for each supported packet type. This isolates the processing logic for a given packet, allowing a Mako to expose any data resource under the Mako protocol. The initial Mako distribution contains a handler implementation to expose XML databases that support the XMLDB API [21]. It also contains handler implementations to expose MakoDB. MakoDB is an XML database optimized for interacting in the Mako framework. Implementations exposing other data services will be distributed as they become available.

5.5 Exposing a Data Service with Mako

Data services can easily be exposed through the Mako protocol by creating an implementation of the abstract handlers. Since there is an abstract handler for each packet type in the protocol, data services can expose all or a subset of the Mako protocol. Once handler implementations exist, Mako can be easily confi gured to use them. This is done in the Mako confi guration fi le, which contains a section for associating each packet type with a handler.

5.6 Global Addressing

We mentioned earlier that Mako provides a method of uniquely identifying elements contained in XML documents. In actuality these elements are uniquely identifi ed across the collection in which they reside. Mako also provides a method of globally addressing XML elements. This is done using the three tuple id, (Mako URI, collection, elementId). Being able to globally address entities within Mako provides several advantages. Most importantly it facilitates data federation across multiple Makos.

5.7 Virtual Inclusion

One example of how data may be federated across multiple Makos is by *virtual inclusion*. Virtual inclusion is a reference within an XML document to another XML document. This means that Mako allows XML documents to be created that may contain references to existing XML documents or elements, both local and remote. In this way, an XML document can be distributed and stored across multiple Mako servers by storing subsections of the document remotely and integrating them with references. This ability is critical in enabling large data documents to be partitioned across a cluster while still maintaining the single document semantics of a model.

5.8 VMako

Utilizing Mako's component architecture, alternate protocol handlers can be installed in a Mako server to enable it to utilize remote Mako instances. The Virtual Mako, illustrated in Figure 3, is a collection of protocol handler implementations that extend the operation of the data services to operate on an administrator-defi ned set of remote Makos. It maps a number of Virtual Collections to a set of remote collections. This simplifi es the client-side complexity of interfacing with multiple Makos by presenting a single virtualized interface to a collection of federated Makos.

For example, the SubmitSchemaHandler is extended to broadcast schema submission requests to all remote Makos, and return an aggregated response. The SubmitXML-Handler utilizes a confi gurable data ingestion algorithm to determine what to do with

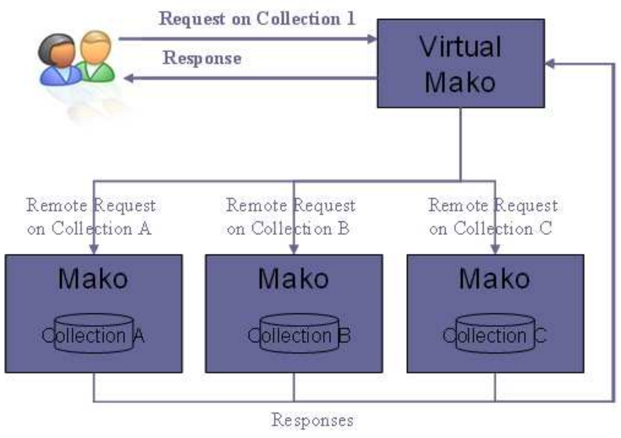

Fig. 3. Virtual Mako

submitted instance data. It can be confi gured to distribute instance data to supporting Makos via a Round Robin scheduler, or any other desired distribution mechanism. Other handlers are implemented in a similar fashion. The Virtual Mako could also be utilized to decluster large data types. By utilizing virtual inclusion, a submitted XML instance document could be broken down into separate sub-documents, and distributed across remote Makos. A new document containing references to these sub-documents would then by created and stored in a selected remote Mako. A XMLDocumentHandle to this new document would be returned to the client. A request to materialize this document would then completely restore the original document.

While this virtualization eases the burden of interfacing with multiple Makos, the primary purpose for a Virtual Mako is to enable distributed query execution. In a Virtual Mako, the XPathHandler is implemented such that requests are broken down into sub-queries and sent to appropriate remote Makos. Responses are then aggregated at the Virtual Mako, and returned to the client. In the current implementation, the Virtual Mako acts as a simple mediator and aggregator. The architecture was designed such that this could be replaced by a sophisticated distributed join implementation without affecting any client-side code.

Future work for extending the Virtual Mako concept will focus on adding semantic information about remote data types such that the Virtual Mako can make informed decisions about where to store data, as well as utilizing this information to enable ad hoc queries. For example, with ontological knowledge of remote data types and their relationships, a query could be executed by utilizing data transformation services and by querying for semantically compatible data types instead of strict class equality.

6 Data Translation Service (DTS)

The Data Translation Service (DTS) is responsible for managing the translation between data types. It accomplishes this task by maintaining a registry of remote mapping services, which provide pairwise translation between namespace elements. As an example, consider the use case where a data type A is required to be translated to datatype *B*. There are two primary motivators for the DTS. The first driver is the need to maintain the link between data stored against a particular version of a schema and future versions, as the schema evolves. As data in Mobius is strongly typed against a particular version of at least one schema, it is important to be able to easily migrate data adhering to a schema when the schema changes. Given a suitable mapping service from one version to the next, data could either be migrated to the new schema, or left unchanged but migrated on-demand when requests for data adhering to the new version is requested.

A second driver is the need to seamlessly translate data types that are semantically similar but syntactically different. As the number of distributed data sources and data types increase, the need for translation between common data types becomes extremely important. It is unreasonable to assume that in a Grid environment all services will use the same data types for services that are slightly related. It is also unreasonable to assume that even if the Grid started out using the same data types for services, that those data types would not slowly evolve and change over time. Inevitably there are subtle differences between services that work on related data that necessitate variations in the data's representation. Rather than try to impose a universal standard representation for all data types, Mobius takes the approach of encouraging organizations to represent their data as they see fit, while still enabling them to inter-operate with similar data types via a DTS published translation service. The hope is that by easing the data transition between separate schema versions, and even completely different schemas, the communities will be able to evolve standards organically where standards are appropriate.

While the DTS enables point to point translation of data types, it is expected that mapping services will be provided that map their data types to appropriate standards, and from said standards to their data types. This will enable a broader range of transitive mappings without requiring explicit mappings for every data type.

The basic DTSR (Data Translation Service Registry) is a simple registry style service architecture. A DTSR will contain the information that describes the individual DTSs that are running on the grid. Any user can build a DTS publishable service as long as their service implements the DTS service interface and registers itself with a DTSR who is responsible for one of the namespaces it is mapping to or from.

Once this service infrastructure is put in place you can envision the DTS being extended to be able to begin to do semantic translation. A user could begin to pose questions like ”Can anyone map my ’car’ to their ’car’?”, where ”car” is some model that has some meaning to the user to be the semantic model of a car. This will require an ontology registry which will store semantic information and relationships about registered namespace elements. It is expected that for many pairs of data types there will be many possible translations between the two, and a semantic model and query language will be required for users to specify translation preferences. Research in this area is ongoing, but it is our belief that a schema-based structural translation service is a necessary base case required prior to supporting semantic translation.

7 Applications

Although Mobius is a research effort still under active development, it has been successfully leveraged as a data management middleware for several applications. In a collaboration with Rescentris, a company focusing on information management solutions to enhance life sciences research and development, Mobius has been utilized to provide a framework for integrating and aggregating collections of disparate medical data sets. This work employed Mobius's ability to store, retrieve, and query distributed data sets modeled by XML schema. By using referenced model elements to describe common medical metadata, this application was able to interrogate biologically relevant portions of distinct medical data sets. Specifi cally, a Virtual Mako was used to query and aggregate distributed data sets representing single nucleotide polymorphisms, and molecular cytogenetic data, via XPath queries.

Mobius has also been leveraged in several applications which operate on large medical image data sets. The Distributed Image Archive System (DPACS), was designed to support generic image archival and management in a grid wide environment, and relied heavily on the Mobius framework's ability to perform on demand data loading using remote element referencing(*Virtual Inclusion*). The DPACS system consists of a front end client application and a collection of federated back end Mako servers. The Mako servers provide the image and metadata storage and retrieval, and the client application can be used by user to upload, query, and inspect the archived data. The DPACS client broadcasts data reference queries to the remote Makos and is able to quickly display the resultant data sets. The requested aspects of the data sets are then loaded on demand as needed by the client, such as the metadata, volumetric thumbnails, and ultimately the image data itself. In another work, Mobius was used to store and manage image processing workflows and image data sets in a system to support rapid implementation and distributed execution of image analysis applications [11], implemented using the Insight Segmentation and Registration Toolkit (ITK) [12] and Visualization Toolkit (VTK) [19]. In this work the processing pipeline discovered work by querying Makos, and processing components utilized Makos to retrieve and store their input and output data sets.

BlockMan, an application which is under current development, supports the distribution and indexing of extremely large data sets which may not fi t within a single nodes storage limit. The BlockMan will ingest large binary data sets, partition and distribute the chunks based on a user specifi ed model and distribution algorithm. It then providesa query interface into data and parallel retrieval of queried results. The uniqueness of this system is its ability of the user to add queryable meaning to the data set by allowing rich structured user-defi ned metadata to be attached to the data chunks. This enables the users to interactively query into datasets which before were not only hard to manage physically but extremely hard to locate and interrogate.

8 Conclusion

As the development of the concept of *the Grid* continues to evolve, we feel protocols and services for managing metadata, data descriptions, and datasets will play a critical role in the ultimate realization of the Grid. In this paper we have introduced the three core services of Mobius: Global Model Exchange (GME), Data Instance Management (Mako), and Data Translation Service (DTS), and shown how they can be leveraged both as a data integration framework in institional research studies, and as key components of a global Grid.

References

1. M. Antonioletti, M. Atkinson, S. Malaika, S. Laws, NW Paton, D. Pearson, and G. Riccardi. Grid Data Service Specification. Technical report, Global Grid Forum, 2003.
2. M. Antonioletti, A. Krause, S. Hastings, S. Langella, S. Malaika, J. Magowan, S. Laws, and NW Paton. Grid Data Service Specification: The XML Realisation. Technical report, Global Grid Forum, 2003.
3. Anders Berglund, Scott Boag (XSL WG), Don Chamberlin, Mary F. Fernndez, Michael Kay, Jonathan Robie, and Jrme Simon. *XML Path Language (XPath)*. World Wide Web Consortium (W3C), 1st edition, August 2003.
4. Data Access and Integration Services. *https://forge.gridforum.org/projects/dais-wg*.
5. I. Foster, C. Kesselman, J.M. Nick, and S. Tuecke. Grid Services for Distributed System Integration. *IEEE*, 35(6):37–46, 2002.
6. I. Foster, C. Kesselman, J.M. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. *http://www.globus.org/ogsa*, 2002.
7. I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. 15(3), 2001.
8. Global Grid Forum. *http://www.gridforum.org/*.
9. The Globus Project. http://www.globus.org.
10. Grid Physics Network (GryPhyN). http://www.griphyn.org/index.php.
11. Shannon Hastings, Tahsin Kurc, Steve Langella, Umit Catalyurek, Tony Pan, and Joel Saltz. Image processing for the grid: A toolkit for building grid-enabled image processing applications. In *CCGrid: IEEE International Symposium on Cluster Computing and the Grid*. IEEE Press, May 2003.
12. National Library of Medicine. Insight Segmentation and Registration Toolkit (ITK). *http://www.itk.org/*.
13. S. Langella. Intellegent data management system. Master's thesis, Computer Science Department, Rensselear Polytechnic Institute, 2002.
14. Andreas Laux and Lars Martin. XUpdate Working Draft. Technical report, http://www.xmldb.org, 2000.
15. Meta Information Catalog (MCAT). http://www.npaci.edu/DICE/SRB/mcat.html.
16. Metadata Catalog Service (MCS). http://www.isi.edu/ deelman/MCS/.
17. Open Grid Services Infrastructure. *https://forge.gridforum.org/projects/ogsi-wg*.
18. Resource Description Framework (RDF). *http://www.w3.org/RDF/*.
19. Will Schroeder, Ken Martin, and Bill Lorensen. *The Visualization Toolkit: An Object-Oriented Approach To 3D Graphics*. Prentice Hall, 2nd edition, 1997.
20. SRB: The Storage Resource Broker. *http://www.npaci.edu/DICE/SRB/index.html*.
21. XML:DB Initiative for XML Databases. *http://www.xmldb.org/*.